Teaching Design Patterns Using Interactive Methods

Mehmet Ata Yurtsever
Bilkent University
matays99@gmail.com

Eray Tuzun
Bilkent University
eraytuzun@cs.bilkent.edu.tr

Abstract

Even though design patterns are one of the most important building blocks in the current software engineering ecosystem, computer science and software engineering graduates face trouble applying these patterns.

To address this, we propose a tutorial and an online lab assessment method to solidify the idea of design patterns for students. The tutorial part integrates a live coding session. The online lab assessment consists of a three-stage process (designing a solution using a class diagram, peer review, and implementation) where students are expected to come up with a fully working solution using design patterns.

The proposed approach is applied twice over two semesters to a total sum of 196 students. We discuss the effects of these interactive educational methods on learning by comparing pre-surveys, post-surveys and analyzing final grades. The analysis of the surveys shows that live coding is highly beneficial in enhancing the understanding of design patterns.

1. Introduction

Design patterns are one of the most prominent concepts in computer science and software engineering. They are frameworks that one can follow to create more concise and overall better code quality [1]. The main idea behind design patterns is to increase code reusability which, in effect, enhances code maintainability, extensibility, and flexibility. Using these patterns also allow teams to use a common vocabulary, allowing them to focus on implementation more [2]. Even though the concept of design patterns is rooted in the work of Christopher Alexander [3], it can be argued that the release of the book Design Patterns: Elements of Reusable Objects [4] is the biggest milestone in its history [5]. Despite their popularity in software engineering practice, design related concepts are not adequately addressed in typical Computer Science or Software Engineering curricula [6, 7]. Previous works defined the need for a more interactive teaching method for design patterns [2, 8].

In this study, we aim to enhance students' understanding of design patterns, using computer science education paradigms such as live coding and peer reviews. We propose a study module that consists of a tutorial and an online lab assignment. In the tutorial, we use live coding and in the lab, we use peer reviews. In live coding, the lecturer designs and creates the project live, in front of the students, which increases student involvement hence enhancing students learning [9, 10]. Research shows that with live coding, students can understand the lecturer's thought process better [9]. In our case, we wanted to show students the process of how to choose the right design patterns for a given problem. This idea is similar to mathematics courses where the lecturer solves example problems for the students. More specifically similar to our case, the lecturer shows how to choose the more appropriate solving method by applying their knowledge to problems. We believe that this would create a better understanding of the underlying motives of design patterns. In the tutorial session, we implemented an arcade game to create a visual example for design patterns. In our game, we included different types of enemies that correspond to strategy design pattern implementations and different extensions of enemies that correspond to decorator design patterns. We incorporated peer reviews into our online lab assignment. Applying peer review techniques into classes generally creates more dynamic and more joyful classes [11]. Peer learning also forces students to think from the lecturers' point of view, allowing them to understand the criteria for a successful work better [12]. Previous work also suggests that students may face anxiety when they review their peers non-anonymously and online peer reviews solve that problem [12].

The rest of the paper is structured as follows. In section two, we discuss previous work concluded in teaching design patterns and solidify our motivation. In

URI: https://hdl.handle.net/10125/79447
978-0-9981331-5-7

section three, we discuss our methodology, going into more detail about the contents of the study. In section four, we discuss our results and in section five, we discuss the threats to validity. Finally, in section six, we present our conclusions.

2. Related Work

One of the early works about design pattern education is published by Wallingford [13]. In his 1996 work, Wallingford [13] mentions his plans on teaching object-oriented design patterns to students. To teach object-oriented design patterns more effectively, there is a need for a good problem decomposition. In a follow-up work in 2000, the author [8] concludes that such decompositions need a more interactive way to be effective. During this time, work by Astrachan et al. [5] defines design patterns as essential to CS curricula. Work by Charles Allison and Neil Harrison [14] shows that the design patterns taught to students are also used later in their courses. They define that learning design patterns as eye-opening for students.

Later work on the design patterns education is mainly focused on syllabus design or better design pattern examples for in-class teaching. Stephen Weiss [15] explains a case study in which he gives simple homework and continues expanding it into a more complex one as the submission deadline approaches. This way students find design patterns themselves naturally. Work concluded by Alphonce et al. [16], mentions three ”killer examples” that explain the power of design patterns thoroughly. In our work, we do not use these examples because they are not closely related to patterns we would like to teach initially. These three ”killer examples” Alphonce et al. [16] mention are chosen from eighteen total ”killer examples” created for workshops over the years. In our examples, we tried to catch the appeal of these ”killer examples”. Our examples are structured after four lessons mentioned in the paper by Alphonce et al. [16]; context, accessibility, real-world, and clear benefits. The example we used in our live coding session is classified as an intra-pattern, but because of constraints such as lecture time and platform, we were unable to apply all intra-pattern lessons mentioned in the paper.

In more recent work, Silva et al. [17] used the Angry Birds game as an example to implement design patterns. They mention that choosing a world-known game reduces the barrier for entry to learn [17]. They conducted a 2-day classroom study which involves students finding usages for mentioned design patterns in the project textually and with diagrams [17]. Students not having enough time between the tutorial and the assessment is one of the mentioned negative comments [17]. In our work, we follow a similar approach to their work, after an interactive tutorial session, we give students more time to study and internalize these subjects. After the gap, we test their knowledge of design patterns not just with textual explanation and diagrams but also by coding.

3. Methodology

Our approach is applied to 196 students over two semesters. The first study is applied to 145 students over 3 sections and the second one is applied to 51 students in a single section. According to the feedback that we got from the first iteration, we updated the format between the studies according to the student feedback. These changes include having a mid-survey and increasing the number of reviews by one. These changes and their effect will be discussed more thoroughly in the following chapters.

3.1. Virtual Classroom Study

A general diagram of our study can be seen in Figure 2. For the rest of the section, we will first give background information on our participants. Then we will mention the tutorial and the online lab in respective order. Finally, we will talk about our measures for the study.

3.1.1. Participants This classroom study was conducted as a part of an Object-Oriented Software Engineering course, where the majority of students were computer science juniors. Prerequisites of the course included two semesters of ‘Algorithms and Programming‘ and ‘Algorithms and Data Structures‘ courses. The Algorithms and Programming course is the introductory course and uses Java; thus we were able to assume all students were proficient in Java. Before our tutorial, the syllabus included ‘Modelling with UML‘, ‘Requirements Elicitation‘, and ‘System Design‘, allowing the tutorial to utilize knowledge from these subjects. The course also includes a term project where students implement board games as groups. Even though it is not a prerequisite, the course is given just after most students’ first mandatory internship. Hence students also had varying degrees of understanding about software engineering.

Table 1. Comparison of How Well Students Understood Concepts.

Iteration	Concept	pre-survey		mid-survey		post-survey	
		Correct	Wrong	Correct	Wrong	Correct	Wrong
1	Strategy Design Pattern	26.6%	73.4%	-	-	58.9%	41.1%
1	Decorator Design Pattern	7.2%	92.8%	-	-	34.7%	65.3%
2	Strategy Design Pattern	21.2%	78.8%	51.2%	48.8%	82.9%	17.1%
2	Decorator Design Pattern	3%	97%	36.4%	63.6%	34.3%	65.7%

Table 2. Comparison of What Students Think About Using These Concepts In Their Final Projects.

Iteration	Question	pre-survey			post-survey		
		Yes	No	Maybe	Yes	No	Maybe
1	Strategy Design Pattern	4.3%	12.2%	83.5%	29.5%	42.1%	28.4%
1	Decorator Design Pattern	1.4%	23%	75.5%	61.1%	11.6%	27.4%
2	Strategy Design Pattern	12%	0%	88%	45.7%	8.6%	45.7%
2	Decorator Design Pattern	9%	0%	91%	25.7%	20%	54.3%

3.1.2. Lecture Lectures were given online and all of the lecture material is also available online[1]. The lectures consisted of two parts; the first part included a small presentation where students were introduced to the idea of code smell and learned the design patterns covered for the lecture, the second part included a live coding session. In both parts, the lecturer explained and exemplified strategy and decorator patterns. These two design patterns were specifically chosen because they are very prominent in game development, making them very effective tools for the term projects of the students. Before this lecture, students had very limited knowledge about the aforementioned patterns and they were not planning on using these patterns in their projects (See Table 1 and Table 2).

The presentation focused on teaching students why and where to use design patterns. After illustrating the connection between code smells and design flaws, the lecture continued with the explanation of the chosen design patterns. UML diagrams were used together with the code implementations. These representations are used comparatively with naive implementations to emphasize their differences. We call a solution or an implementation naive if it does not include any design patterns and it is a direct approach to the problem. For the presentation part of the lecture, we used abstract examples that are common in most books on the subject such as [18]. Our example was a boombox which we strategized and decorated how it plays.

The live coding session consisted of implementing a range of features for a simple arcade game. The proposed game has a character that users can move and types of enemies such that each enemy type moves differently. In the game, the player tries to escape from different enemies by moving the character in a two-dimensional space. Going to the session, there was some initial code that defined a movable character, the main program, and an interface for the enemy class. The proposed enemy class includes a render method that renders the enemy in the correct position and an update method that updates its position. Additionally, the Enemy class also includes its vertical and horizontal speeds as fields. For the implementation, the Processing[2] library and the Processing IDE were used. The Processing IDE has an intuitive interface that is easy to grasp and the Processing library has simple tools for graphical software development. Also, we used processing for Java because students were proficient in Java as mentioned before and Java is an appropriate language to show design patterns due to its many object-oriented features.

The live coding session was split into two phases: strategy pattern implementation and decorator pattern implementation [4]. Our aim for live coding is to show points of interest for design pattern applications. To point this out, we have chosen problems that have pitfalls in their naive solution. In each phase, the session follows through a naive approach until students start finding code smells. This way, students found the points of interest for design patterns themselves. In some sections, this occurred at the very end and required the lecturer to ask; but in other sections, students were able to identify the design flaws earlier. When a design flaw was found, the whole classroom tried to apply the knowledge they accumulated earlier in the lecture to solve it. This allowed students to be more active in the class and use the freshly acquired knowledge. To emphasize why to use design patterns, a UML diagram for the current state of the program accompanied the code. This way students were able to see the bigger

[1] https://osf.io/s3dvt/?view_only=fa7c7939a6b44cf3b7ba87c847ffef59

[2] https://processing.org

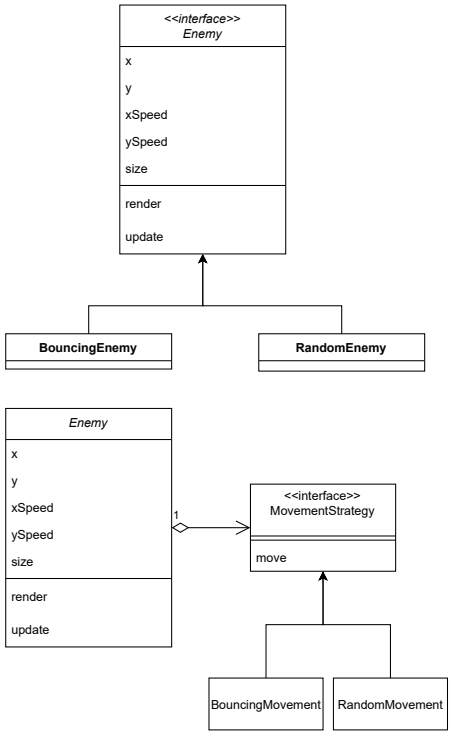

Figure 1. Naive (top) and proposed (bottom) solutions for the first phase of live coding.

picture and understand established code reuse as well as how simpler the code is in this new state.

The first phase of the live coding is about the strategy design pattern. The strategy implementation was chosen over the decorator because it is easier to understand and it is easier to implement decorators over strategies compared to the other way around. The strategy design pattern problem required the design and implementation of two types of enemies. The bouncing enemy bounces when it hits a side of the screen and the random enemy which randomizes its speed when it hits the sides. It could be observed that the only difference between these two enemies is their update method. Both classes have the same render function and constructor, making them great candidates for strategy design pattern application. A naive solution for this is an implementation of these two classes as separate implementations of the Enemy interface as shown in Figure 1. First, the bouncing enemy is implemented followed by the random enemy. During this part, it is expected for students to find smells when going from bouncing enemy to random enemy. To emphasize that, following the bouncing enemy implementation the bouncing enemy class is copied and pasted to a new file and renamed. During this process, emphasizing that they are not very different is another good nudge for students. Usage of the strategy design pattern is not the only perfect solution for the situation. Same code reuse is achievable by creating an abstract Enemy class that has an abstract update method and implementing these two as its children. This creates an interesting discussion topic in the middle of the lecture because both solutions have their pros and cons.

The second phase of the live coding is about the decorator pattern. There are two decorators that apply to both strategies planned for this stage. The first one, the heavy movement decorator, is known from the start while the second one, the right movement decorator is mentioned later. Heavy movement decorator increments the vertical speed field of the enemy gradually, creating a feeling of gravity. The right decorator is very similar to the heavy decorator, the only difference being the axis that the speed is incremented. Similar to the previous phase, the implementation starts on the wrong track by defining the heavy movement decorator for each strategy using inheritance. Making this in the same file promotes students to see the duplicate code. In this case, the logic for movement is passed by inheritance. In this phase, students are expected to find the code smell when moving from the first decorator implementation to the next. After the discussion on the naive implementation, the decorator design pattern is implemented. Then a quick discussion on decorators is initiated with students. Later, right movement decorator is mentioned and implemented using the decorator design pattern. The reason to include the right decorator is to show combined decorators. Without the decorator design pattern, implementation of these decorated movements would have taken six different classes but with the decorator pattern, it only took two extra classes.

3.1.3. Online Lab Assignment A special platform for the lab is created which is explained in more detail later in this section. The assignment used the power of peer assessment. Before the lab, the system pairs students anonymously in order of the given student list. Randomness is achieved by randomizing this file beforehand. Research shows that anonymous peers result in better reviews [12]. The lab consisted of three stages; named design, review, and coding as they are respectively shown in Figure 2.

There was two different problems in each instance of our study. Each student solves one question and reviews the other. This way each student will face both problems either by solving or reviewing. Before the lab assignment, each student gets a unique account to use on the platform. Phase switches are simultaneous for each student, meaning students cannot finish the first stages

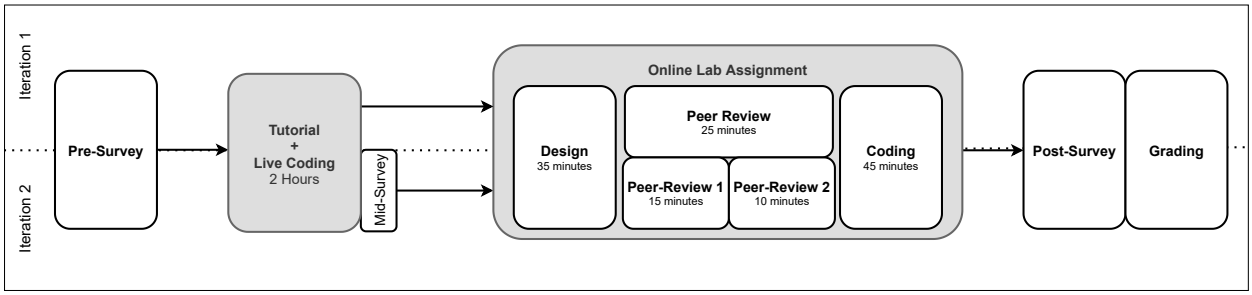

Figure 2. Flow of the study

early and pass onto the next stage. This simultaneous change is controllable from an admin panel.

During the design stage, students are expected to design solutions for problems. These problems are presented as company assignments to imitate real life and available online[3]. In both of the problems, it starts with the basic explanation of the current system, continues by mentioning the requirement, and finishes by giving example usages. They are expected to draw a UML class diagram, upload it, and create an explanation. This way, the lab can check the students' knowledge of design patterns as well as UML diagrams.

After the design phase, students directly pass onto the review phase. Students can see the other problem and their peer's solution to it in the review phase, including the text and the diagram. They are expected to write a short paragraph on the errors or good points of their peer's review. In our second instance of the study, students review solutions of two other student allowing each student to get two reviews from different perspectives. Because students review the same question twice, we were able to allocate less time for the second reviews phase.

Finally, in the coding phase, students can see their original problem, their own solution to it, and the review they got from their peers. They are expected to reflect on the review and make all the necessary changes to come up with a Java implementation in the coding editor on the right as shown in Figure 3. This phase is the longest in the assignment. The extra time enables students more time to reflect on their errors resulting in better learning [19].

3.1.4. Measures Before the classroom study, students are asked to fill a pre-survey that checks their prior design patterns' knowledge. After the study, we also asked students to fill a more thorough post-survey. For our second instance of study, we asked for a mid-survey just after the tutorial. All surveys include multiple-choice questions and rating scale questions, the rating scale is between 1 and 5 for each question, also the post-survey included open-ended questions for them to leave some extra remarks such as in Figure 4.

This study was a graded assignment for the course. It was worth 7% of the course total. Students are graded according to their work in the online lab assignment. Each phase of the lab assignment was graded separately. Design, review, and coding sections were graded as 40%, 15%, and 45% (35%, 30%, and 35% for the second instance of the study) of the total grade respectively. We have changed our grading criteria in the second iteration because we tried to suggest our students to give better reviews. After the study, students are asked to fill a post-survey which includes questions for feedback on the lab and illustrates the information they have acquired since the start of the study.

3.1.5. Deliverables Deliverables in the study can be grouped into phases that they belong to. In the design phase, we expect a UML diagram and a short paragraph of text. From these deliverables we expect the student to apply any design pattern (it does not matter if it is the correct one), explain it clearly, and draw a UML class diagram that represents it correctly. This way, we can check if students are able to design solutions to problems that they face. Having a correct solution is graded very minimally in these deliverables because just as in real life student has more time and feedback ahead to find the best solution.

In the peer review phase, we expect students to point out mistakes (or correct points if the reviewed solution is entirely correct). These mistakes can range from simple UML errors to important design choices. We value each feedback that even just point out small mistakes. Students can lose all the points in this phase if they give wrong or misleading feedback. As we mentioned earlier, we explicitly stated this, hence some students refrained from giving feedback that they were not sure of. In this phase, students can also lose points if they miss obvious

[3] https://osf.io/s3dvt/?view_only=fa7c7939a6b44cf3b7ba87c847ffef59

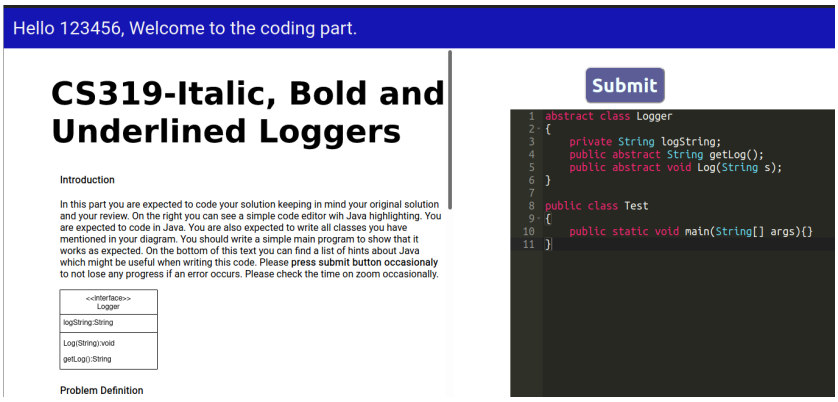

Figure 3. The coding page from the platform

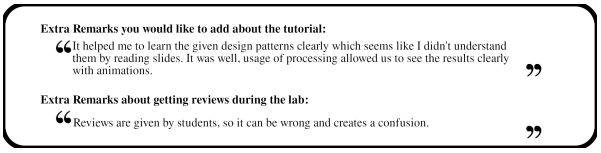

Figure 4. Some example extra remarks from the post-survey

errors in their peer's solution. A good example for a review from our second iteration can be found below.

> As far as I am concerned, the main idea of the solution is correct but generally it is shown wrong. Firstly, I think it should be the decorator design pattern rather than the strategy. Secondly, inheritances between "BorderStrategy" and sub-strategies are missing. I am not sure whether we should show the constructor but since I believe it should be decorator and decorator design pattern should get parameter, there should be constructors in sub-strategy classes. On the other hand, "Canvas" is abstract which is correct, and there a "SimpleCanvas" class that inherits it. Also, aggregation and its relation is shown which is a plus.

In the coding phase, we expect students to find the best design pattern for the problem and create a working and simple solution using Java. We also check whether the student can use object-oriented features such as inheritance, interfaces, and aggregation to solve design problems. The student can get some points by demonstrating their ability to use these features even if they are unable to come up with the best solution.

3.2. Platform

The platform for the online lab assignment is created specifically for the study. It allows students to log in using the account given to them. The platform has a single page accessible by the students. This page changes according to the phase. There is also an admin account that allows proctors to add accounts, analyze solutions, and change phases manually. When adding accounts, the system automatically pairs students and assigns them their problems. A back-end system stores student' data using postgreSQL. Admin can also list all solutions on the system. The whole system is written in Java using spring-boot[4] and Thymeleaf[5]. By default, the system starts in a wait phase which shows the waiting message to students when they enter the system and blocks all updates to the database.

Proctor can switch to the design phase using the control panel. When switching between states, students need to refresh their pages to get to the new phase's page. In the design phase and all other phases, the screen is divided into two parts. The left part shows the definition of the problem, expectations, and examples. The right of the screen includes necessary text boxes and buttons for submission; such as a file upload button, textbox, and submit button. All of the submissions create a popup message on the bottom right of the screen. This popup can be green or red depending on the submission status.

Proctor again can switch to the review phase using the control panel. Students on the design phase page can no longer submit their solutions because they are in a different phase. When students refresh they are greeted with the review screen. The review screen includes their peer's problem and solution on the left, also a textbox for the review on the right. The coding screen includes

[4]https://spring.io/projects/spring-boot
[5]https://www.thymeleaf.org/

the student's own problem, solution, and review on the left and a code editor on the right as shown in Figure 3.

4. Results

In this section, we discuss the data collected by our surveys and the results of the lab. In the first iteration, the post-survey was filled by 95 students, in the second one, the post-survey was filled by 35 students. The percentages discussed consider both iterations.

4.1. Perceived Effectiveness of the Tutorial

According to the post-survey results, the students found the usage of live coding in the tutorials very helpful, 86% of the students rating it 3/5 or more (see Figure 5). Only 7.2% of the students believed that live coding was very unhelpful. In the extra remarks, students mentioned that usage of Processing and creating a graphical example for the subject was helpful and made it easier to see the results.

Only 5.6% of the students believed that the tutorial was very unhelpful (see Figure 5). Overall 85% of the students rated it 3/5 or more. In extra remarks, students mentioned that the tutorial was structured in a way that helped them understand the subject they were unable to before. Student feedback mentioned that this tutorial helped them understand design patterns better compared to just reading from slides.

In our second iteration of the study, we applied a mid-survey just after the tutorial. This way we can compare student answers and find the most important element in our study. From the results, we see that tutorial helped the confidence of the students in using design patterns. The tutorial helped explain decorator design patterns but most students learned about strategy design patterns after the lab.

4.2. Effectiveness of the Online Lab

Problems in the lab were in general well received. Most of the students believed that problems were connected to the tutorial, 86% of the students rating it 3/5 or more (see Figure 7). And almost no students found the problems very easy or very difficult. Sixty-two percent of the students from the first iteration voted that problems were average in complexity (see Figure 6). We can see that answers for the second iteration have a bit more variance. Also average for the lab is a bit lower.

We got different results for different iterations on the subject of the reviews. In the first iteration, students believed that getting reviews were not very helpful (see Figure 5). From the extra remarks section, we can

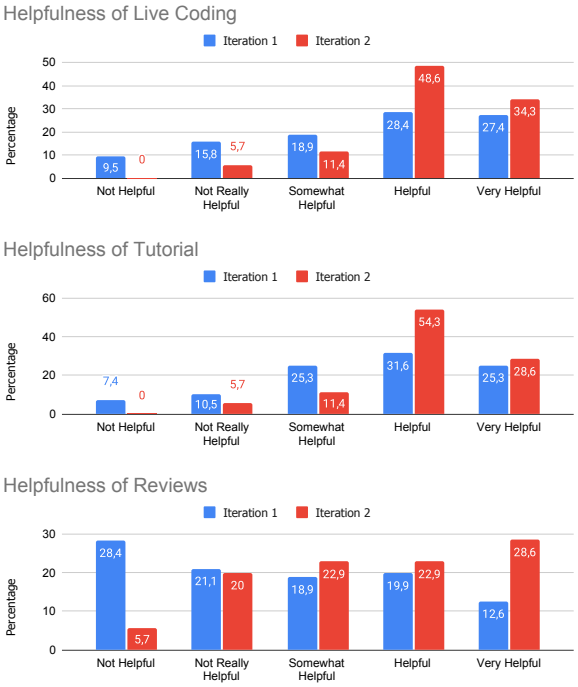

Figure 5. Helpfulness of the study elements

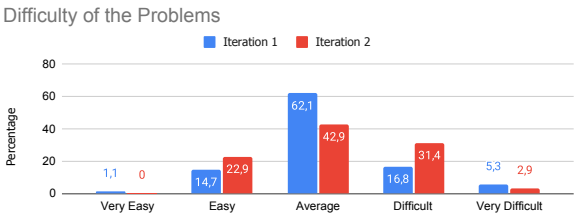

Figure 6. Difficulty of the problems

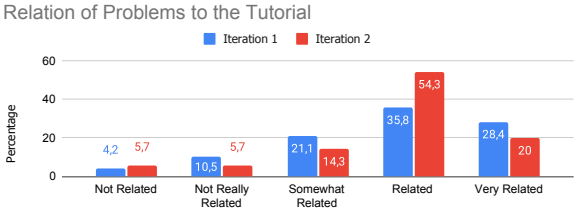

Figure 7. Problems' relation to the tutorial

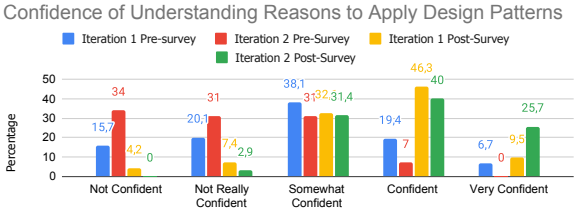

Figure 8. Students Confidence on understanding the reasons to apply design patterns

see that this is because they either did not believe in their peer's review or they believed that their peer was rude as they mention in extra remarks. Such extra remarks can be seen in Figure 4. This showed that the review phase needed more explanation. Hence, we applied two review processes for our second iteration. We also observe that students having a good review have minimal impact on their peer's coding phase grade. In the first iteration, if we remove students with perfect scores from the solution phase, the average for the coding phase is 30.15/45, and the students who got perfect reviews average only 30.33/45 which is a very minimal difference. On the contrary, in the second iteration students found reviews fairly useful (see Figure 5). Also according to the post-survey, 52.9% of the students had their mistakes pointed out to them. Probably the most important reason is, that students got two different reviews they could rely on them more.

4.3. Overall Effectiveness of Study

Comparing pre-surveys to post-surveys, we can see that after our study, students are more confident hence more likely to apply these patterns to their projects (see Figure 8. In our study students were already somewhat confident but in our second study, almost no student mentioned that they were confident in using design patterns. We can see that both groups ended up in a similar and definitely a better spot after the study. We can also conclude that study excels at teaching strategy design patterns to the students, especially in our second study results show that almost every student had a clear understanding of the subject. In both instances, all students had more trouble understanding the decorator design pattern compared to the strategy pattern. Probably because the concept of decorator design pattern is more complex.

5. Threats to Validity

There was a two-week gap between the tutorial and the online lab assessment. During these two weeks, other lectures on design patterns continued. This way students learned more about design patterns outside the scope of this study. Unfortunately, since we do not have data in a similar format from previous years, it is difficult to conclude a direct benefit of the study.

Previous works on design patterns education aimed to incorporate design patterns into novice programming courses while this study was conducted on junior computer science students. Thus, we cannot claim whether similar results can be achieved with more novice programmers.

Because the study was a graded assessment, we were unable to create control groups that do not take part in live coding session or peer review. This makes our result prone to errors. To counteract this, we tried to make our conclusions over the students' feedback as much as possible.

The second iteration of the study was concluded on a smaller group, resulting in fewer data to work with. Also we can see that attendance was also lower for pre survey in the second iteration. Pre and post surveys were filled by 92% and 63% in the first iteration while in the second iteration they were filled by the 70% of the class each time (which was the attendance to the tutorial). All the surveys were voluntary, thus they may not represent the whole class. In the second iteration we changed the structure such that students reviewed two other solutions, hence some differences might be caused by that.

Because the platform was online, it was prone to cheating. We took precautions for such behaviours by keeping everyone in the same zoom call with cameras open. Even though we would like to believe that no such behaviour occurred, plagiarism might have affected our results.

6. Conclusion

In this study, we conducted a virtual classroom study for the junior Computer Science students by suggesting a different way of teaching design patterns. We formed a unique approach by using the previously researched methods in computer science: peer reviews and live coding. In the study, we discussed the strategy and the decorator design patterns. The study was split into two parts. The first part included a two-hour lecture with live coding while the second part was a graded online lab assignment that included a peer review component.

From the student survey, we can see that the lecture was useful to the students. The live coding and the usage of graphical examples helped students understand concepts better. With live coding, they can understand the thought processes behind the code better. Also having a game that they can interact with, later on, may increase the number of students revising the class material after the course.

The online lab assignment consisted of three phases; design of the solution, peer review, and coding of the solution. During the lab assignment, each student started each phase synchronously. The review part required students to perform peer reviews. Results show that when the study is concluded with a single review for each student they had minimal effect on students' results. A reason for this might be the misleading reviews where a student makes wrong suggestions to

a solution provided by peers. The possibility of such reviews harmed the trust for the reviews. When we inspect solutions individually, we can see many students carried on with their wrong solutions even if they had a great solution in the review. With this result, in the second iteration, the lab assignment was altered such that students received two reviews instead of one. With the improved peer review process, more than half of the students found out mistakes in their solutions.

In summary, we can see that our approach was efficient to teach the design patterns. Students had a chance to understand and apply the design patterns simultaneously. Choosing patterns directly tied to their current goals helped students connect with the subject more.

We shared the lecture slides, and assignment materials publicly to make them accessible for the Computer Science education community. In the future, we are planning to add more design patterns to expand the scope of the tutorial.

References

[1] D. Nguyen and S. B. Wong, "Patterns for decoupling data structures and algorithms," *SIGCSE Bull.*, vol. 31, p. 87–91, Mar. 1999.

[2] B. Goldfedder and L. Rising, "A training experience with patterns," *Commun. ACM*, vol. 39, p. 60–64, Oct. 1996.

[3] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language*. Oxford University Press, 1977.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Elements of Reusable Software," *Addison-Wesley Professional Computing Series*, 1996.

[5] O. Astrachan, G. Mitchener, G. Berry, and L. Cox, "Design patterns: An essential component of cs curricula," *SIGCSE Bull.*, vol. 30, p. 153–160, Mar. 1998.

[6] V. Garousi, G. Giray, E. Tüzün, C. Catal, and M. Felderer, "Aligning software engineering education with industrial needs: a meta-analysis," *Journal of Systems and Software*, vol. 156, pp. 65–83, 2019.

[7] V. Garousi, G. Giray, and E. Tuzun, "Understanding the knowledge gaps of software engineers: An empirical analysis based on swebok," *ACM Transactions on Computing Education (TOCE)*, vol. 20, no. 1, pp. 1–33, 2019.

[8] E. Wallingford, "Using patterns in the CS curriculum," in *Proceedings of the Fifth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges*, CCSC '00, (Evansville, IN, USA), p. 235–237, Consortium for Computing Sciences in Colleges, 2000.

[9] M. J. Rubin, "The effectiveness of live-coding to teach introductory programming," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, (New York, NY, USA), p. 651–656, Association for Computing Machinery, 2013.

[10] A. G. S. Raj, J. M. Patel, R. Halverson, and E. R. Halverson, "Role of live-coding in learning introductory programming," in *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*, Koli Calling '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[11] G. Petonito, "Fostering Peer Learning in the College Classroom," *Teaching Sociology*, vol. 19, p. 498, oct 1991.

[12] X.-M. Wang, G.-J. Hwang, Z.-Y. Liang, and H.-Y. Wang, "Enhancing students' computer programming performances, critical thinking awareness and attitudes towards programming: An online peerassessment attempt," *Educational Technology and Society*, vol. 20, pp. 58–68, 01 2017.

[13] E. Wallingford, "Toward a first course based on object-oriented patterns," *SIGCSE Bull.*, vol. 28, p. 27–31, Mar. 1996.

[14] C. D. Allison and N. B. Harrison, "Teaching design patterns: A matter of principle," *J. Comput. Sci. Coll.*, vol. 23, p. 206–211, Oct. 2007.

[15] S. Weiss, "Teaching design patterns by stealth," *SIGCSE Bull.*, vol. 37, p. 492–494, Feb. 2005.

[16] C. Alphonce, M. Caspersen, and A. Decker, "Killer "killer examples" for design patterns," *SIGCSE Bull.*, vol. 39, p. 228–232, Mar. 2007.

[17] D. S. da Cruz Silva, M. Schots, and L. Duboc, "Fostering design patterns education: An exemplar inspired in the angry birds game franchise," in *Proceedings of the XVIII Brazilian Symposium on Software Quality*, SBQS'19, (New York, NY, USA), p. 168–177, Association for Computing Machinery, 2019.

[18] E. Freeman, E. Freeman, B. Bates, and K. Sierra, *Head First Design Patterns*. O' Reilly Associates, Inc., 2004.

[19] S. Einarsdóttir, L. Hearne, M. Rammo, P. Weber, and F. Wikstrand, "Peer Learning for the Quality Enhancement of Degree Programmes," in *European Competence Standards for the Academic Training of Career Practitioners*, pp. 89–106, Verlag Barbara Budrich, Mar 2019.